Time-Space Trade-offs for Computing (High Order) Voronoi Diagrams

Matias Korman

Joint with: Bahareh Banyassady, Wolfgang Mulzer, André van Renssen, Marcel Roeloffzen, Paul Seiferth, and Yannik Stein

Memory Constrained Model

- Pick your favourite problem
- Best algorithm runs in T time using S space
- What can you do if you do **not** have S space?
- Alternative algorithm for S'<S space and T'>T time
- What if your space lies in between?

Memory Constrained Model

- **Input** is in a **read-only** data structure
  - Streaming/Multi-pass: one/O(1) scans of input allowed
  - **Sequential/Random Access**: unlimited scans, never modify
  - In-place: reorder/modifying input is also possible
- We write **output** in a **write-only** list
  - Once written, we cannot read, modify or delete
  - Must output solution in order

Memory Constrained Model

- We can use **s variables** in addition to the input
  - Each variable stores a number/pointer or similar info ($\theta(\log n)$ bits)
  - Parameter determined by the user (smaller than the input)
  - Implicit storage consumption (i.e., recursion) is also counted
- **Aim:** algorithm whose running time that decreases as s grows
  - i.e., $O(n^2/s)$ time using $O(s)$ variables
  - In most cases the time-space dependency is linear

Why?

Started in the late 60's when memory was expensive

Renewed interest with distributed computing/handheld device

- Embedded systems
- Ad-hoc networks

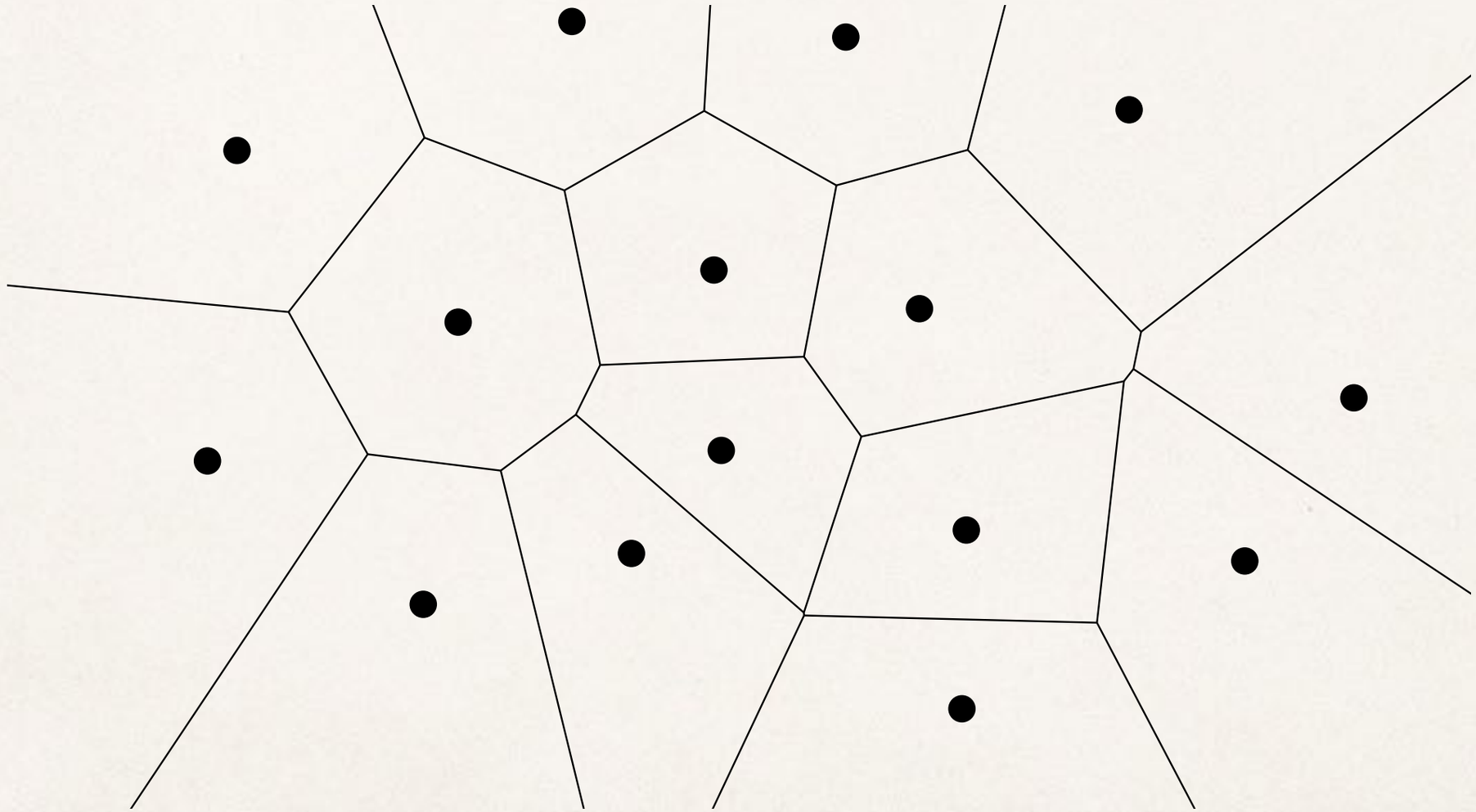
Sometimes we need restrict memory usage and data modification

- Concurrent programs
- Privacy issues

Why?

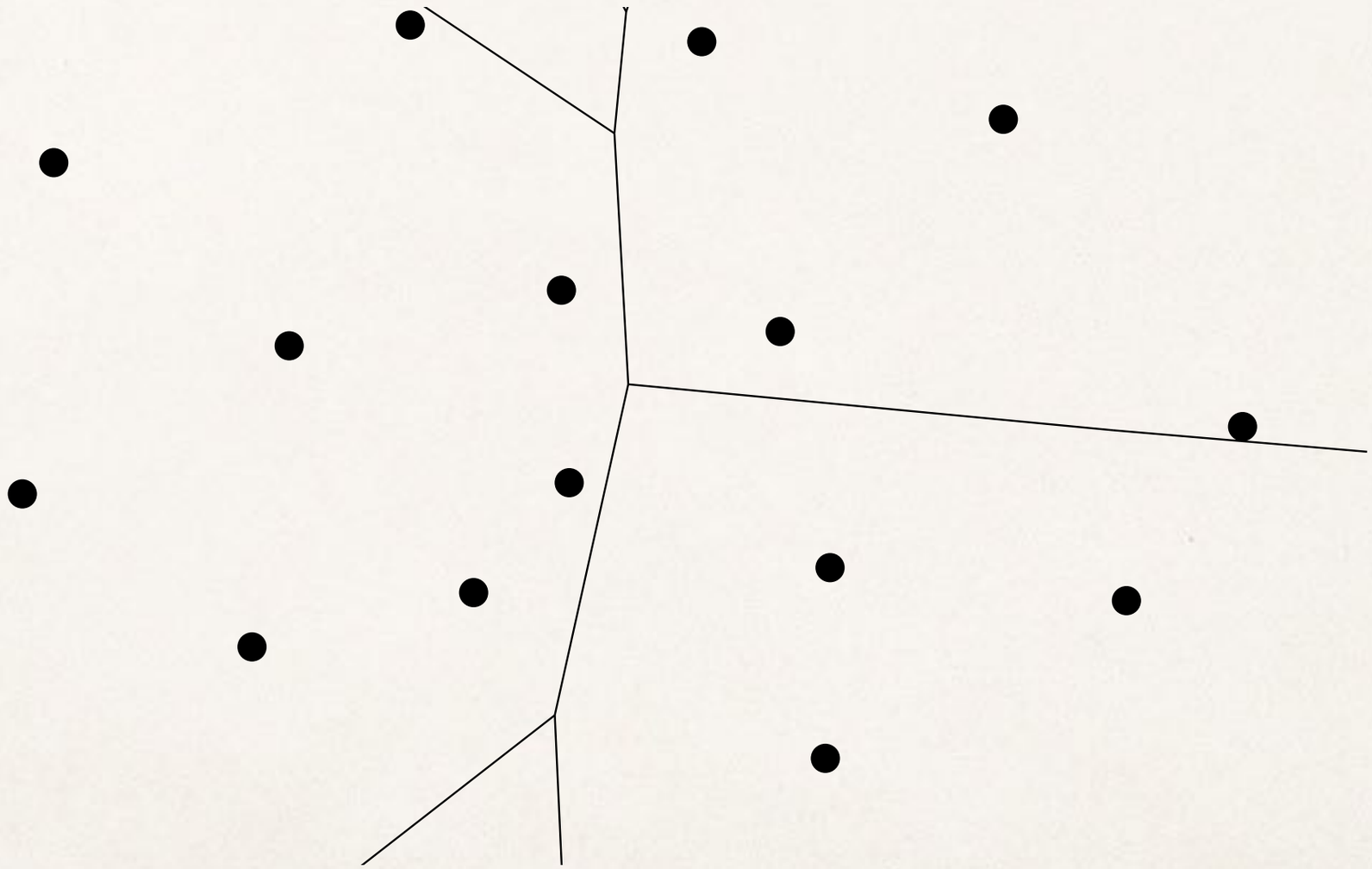
It is a cool problem!!

Voronoi Diagram



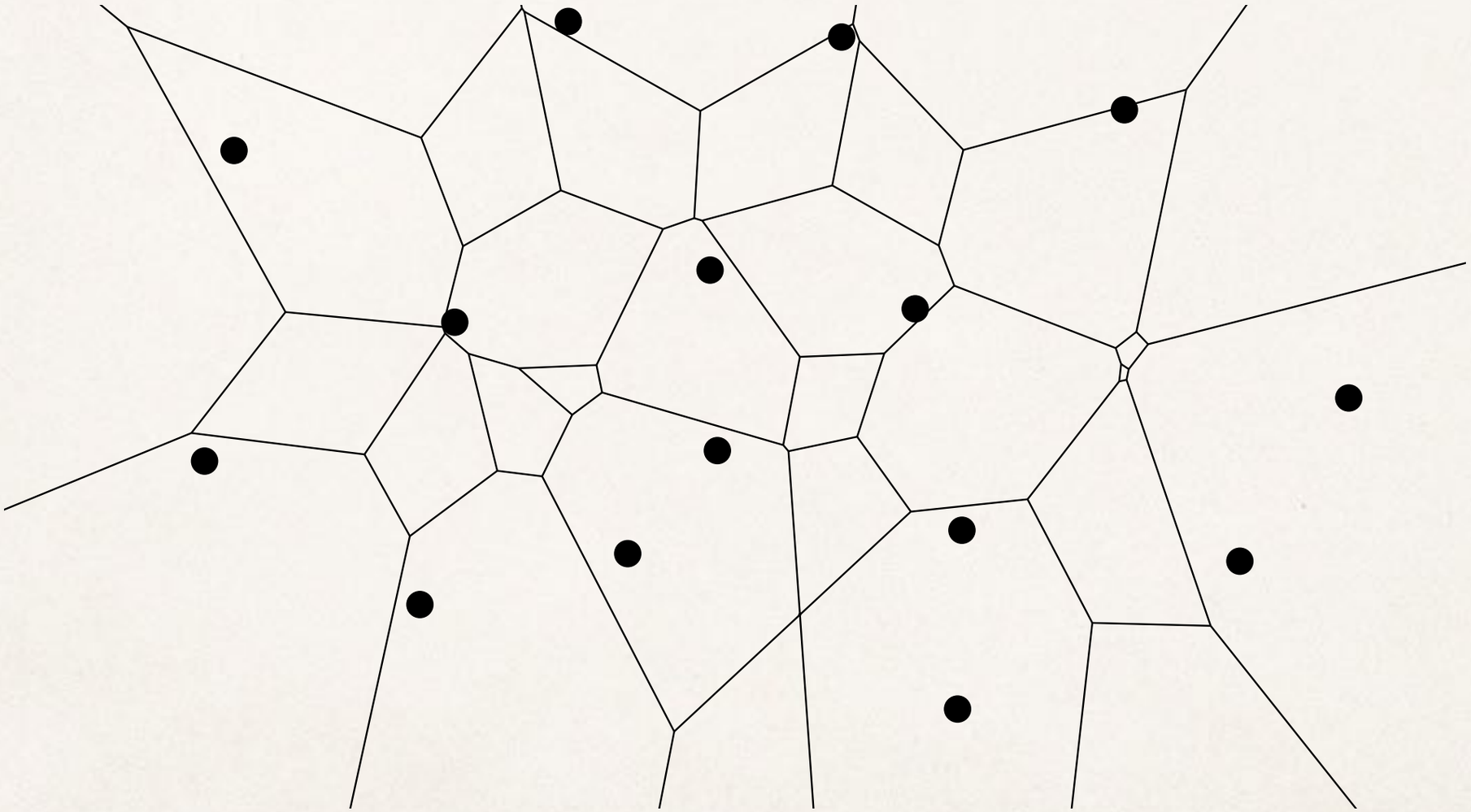
- Partition of the plane into cells with the same **nearest** site

Farthest Voronoi Diagram



- Partition of the plane into cells with the same **farthest** site

Order k Voronoi Diagram



- Partition of the plane into cells with the same **set of k nearest** sites

Problem Definition

- Can we compute the order k voronoi diagram of n sites?
- How fast?
- How much memory is needed?
- Can we make a time-space trade-off?

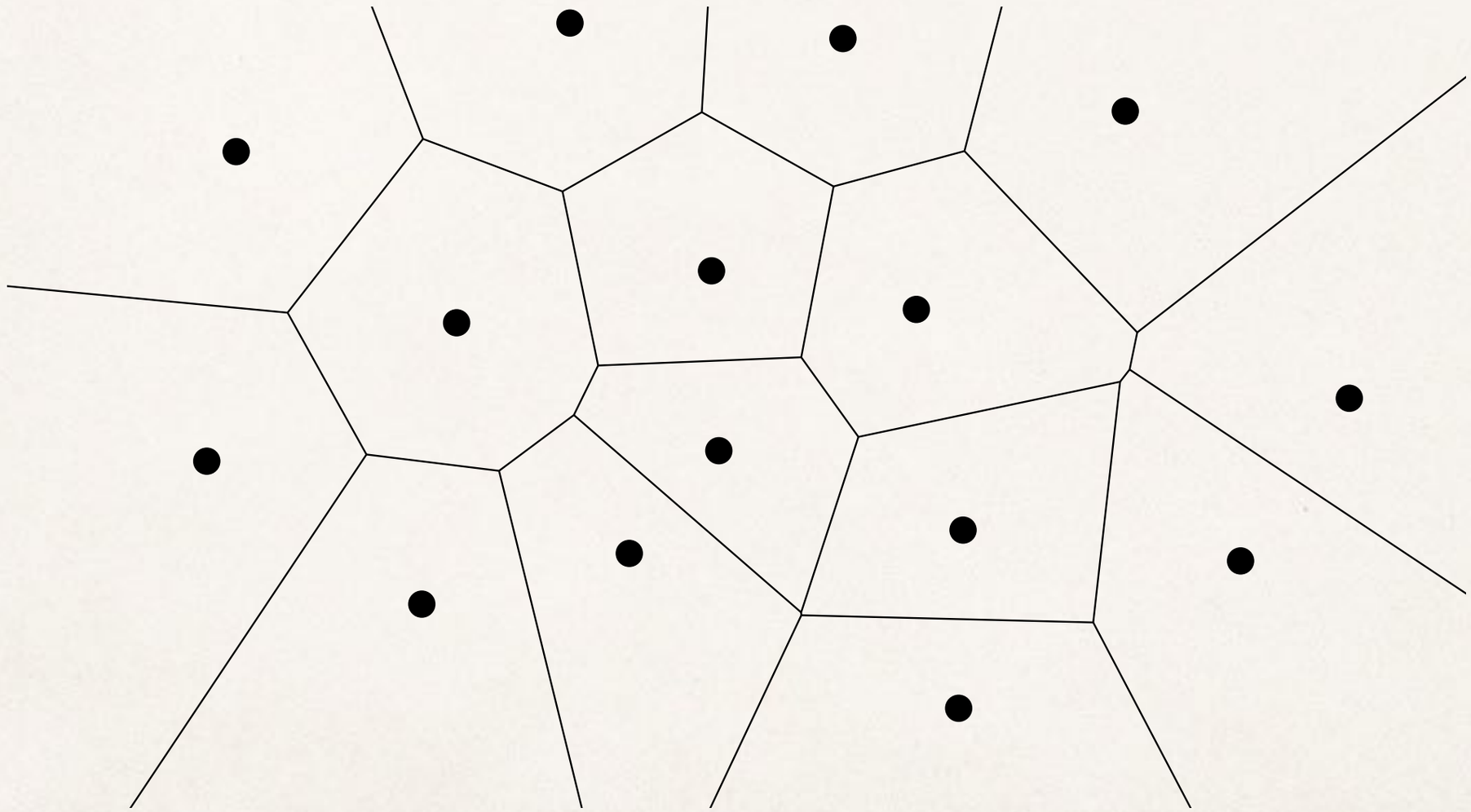
Previous Work

- Unconstrained setting
  - O(nlogn) time for k=1 and k=n-1
  - O(nk polylog(k,n)) for general k
- Memory Constrained
  - $O(n^2)$ time using O(1) words (k=1)
  - $O((n^2/s) \log s + n \log n \log^* n)$ expected time, O(s) words, k=1

Our Contribution

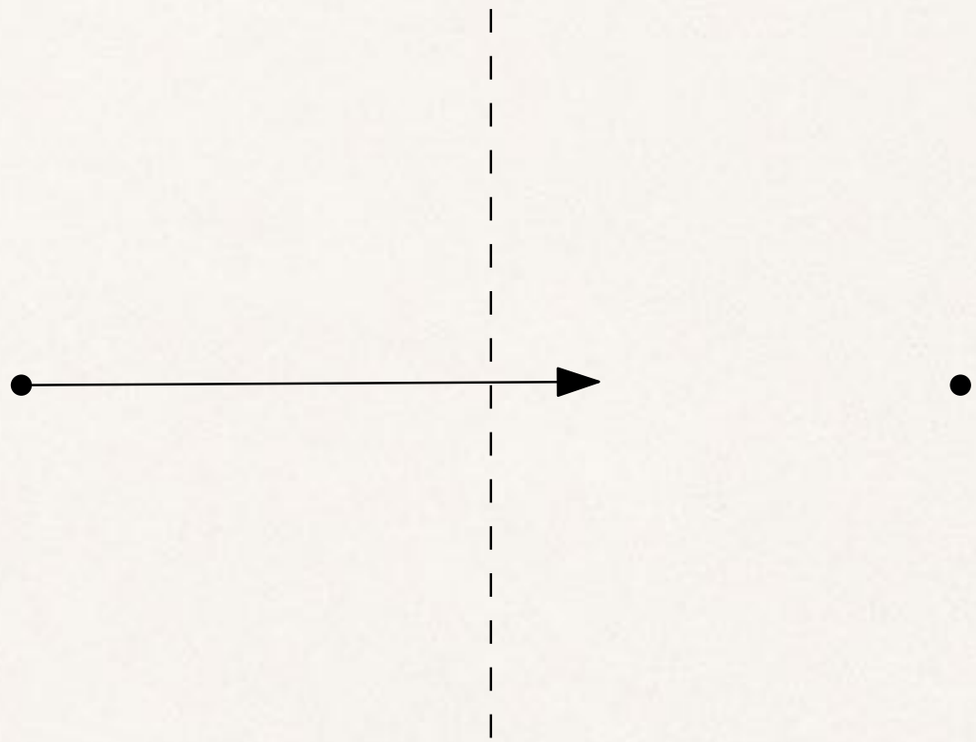
- Unconstrained setting
  - O(nlogn) time for k=1 and k=n-1
  - O(nk polylog(k,n)) for general k
- Memory Constrained
  - $O(n^2)$ time using O(1) words (k=1 **or k=n-1**)
  - $O((n^2/s) \log s + n \log s \log^* s)$ ~~expected~~ time, O(s) words, k=1 **or n-1**
  - **$O((n^2k^6/s)$ polylog(k,s)) using O(s) words for $k<\sqrt{s}$**

Algorithm for O(1) space



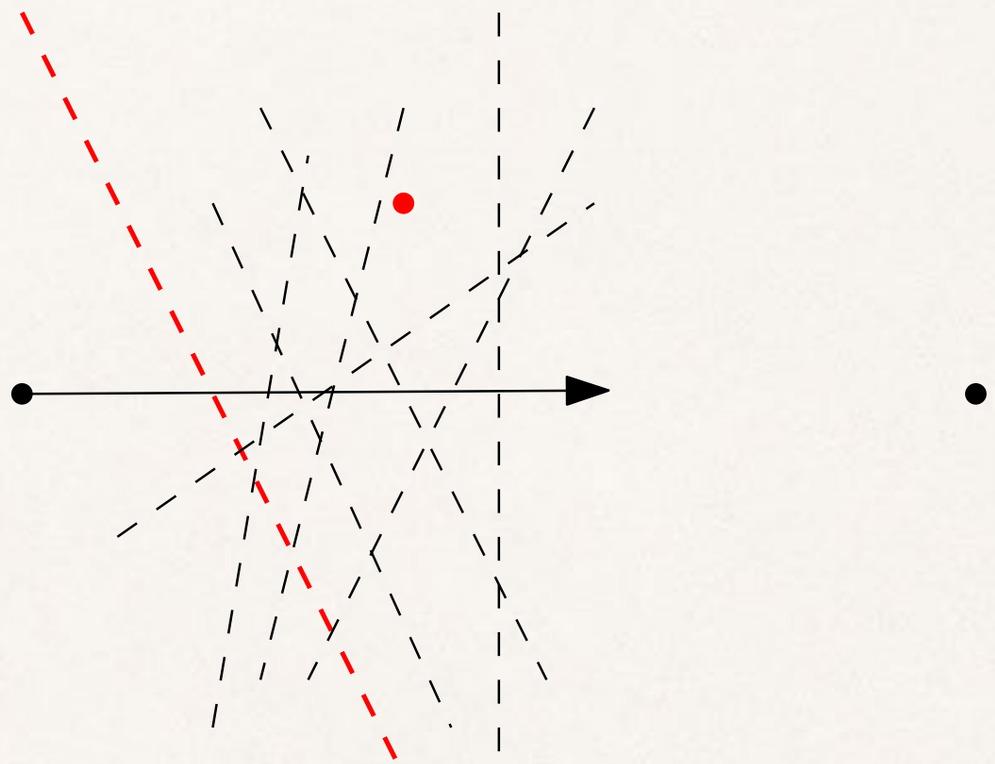
How can we compute the diagram with O(1) words?

Algorithm idea: walk the boundary



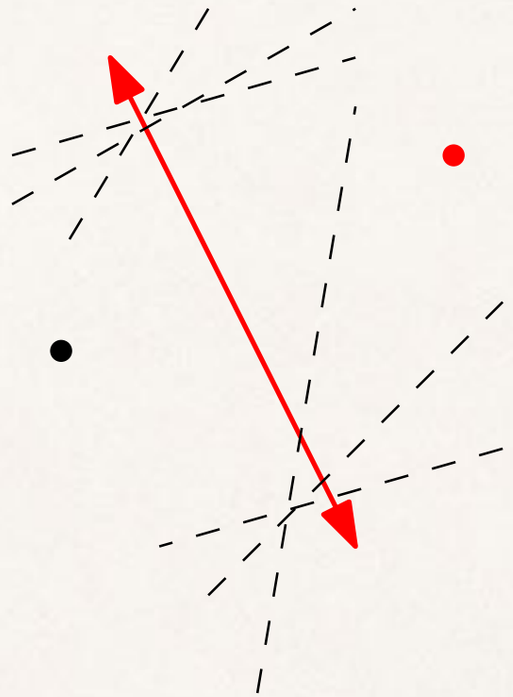
- Shoot a ray and look at bisectors
  - Closest one will be in the diagram
- Repeat once more to find the endpoints

Algorithm idea: walk the boundary



- Shoot a ray and look at bisectors
  - Closest one will be in the diagram
- Repeat once more to find the endpoints

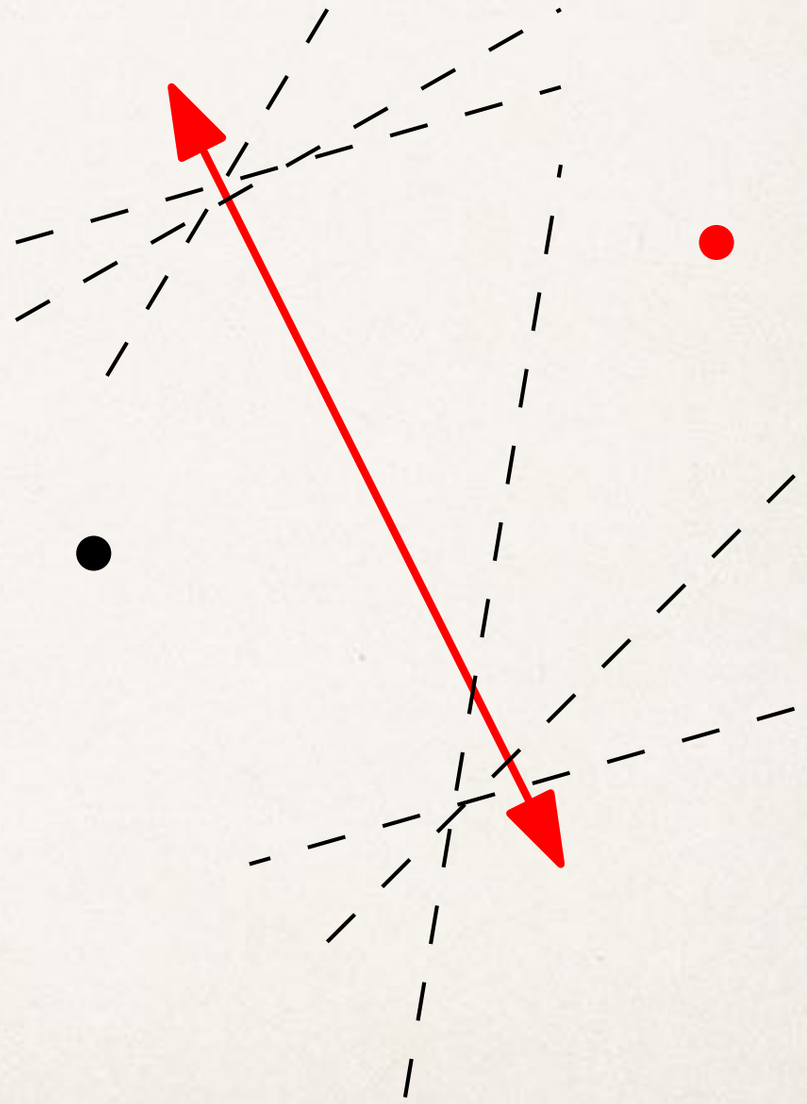
Algorithm idea: walk the boundary



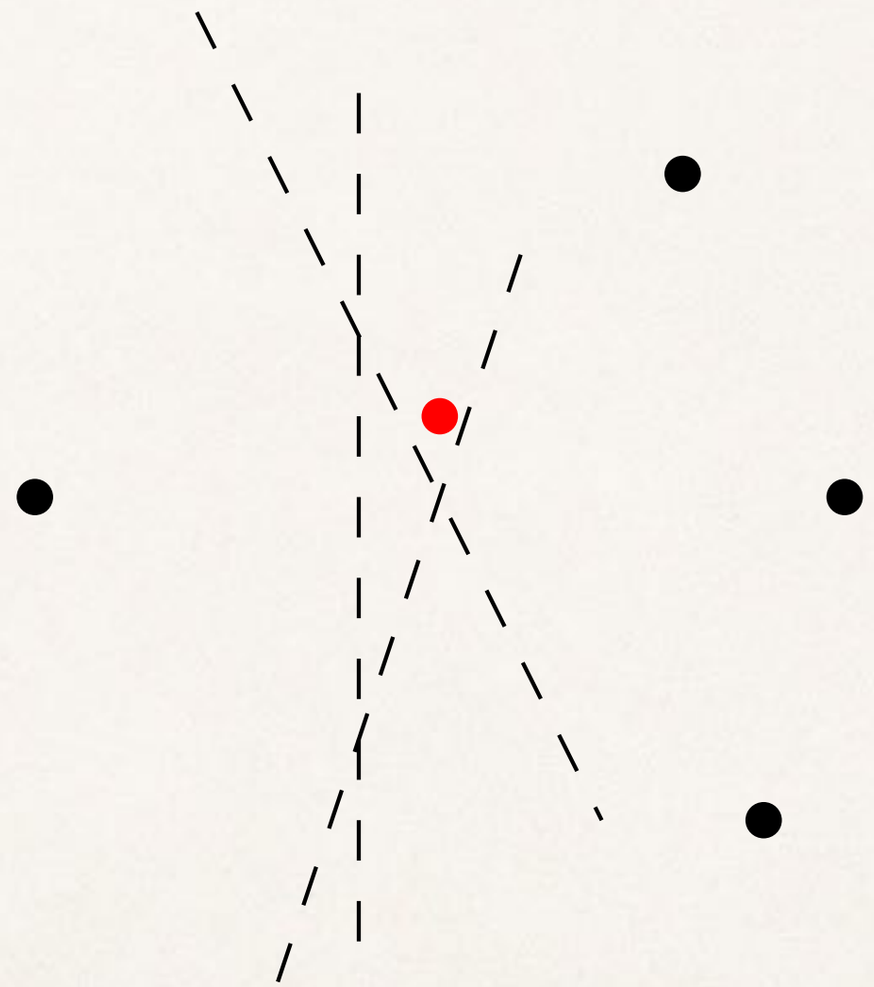
- Shoot a ray and look at bisectors
  - Closest one will be in the diagram
- Repeat once more to find the endpoints

Algorithm using O(1) words

- Two scans of the input, O(n) time
  - Rotate the ray clockwise and repeat
    - Need a *good* starting direction
- If face of site is computed, grab a new site
- Every two scans we find an edge
  - O(n) edges in total → $O(n^2)$ runtime

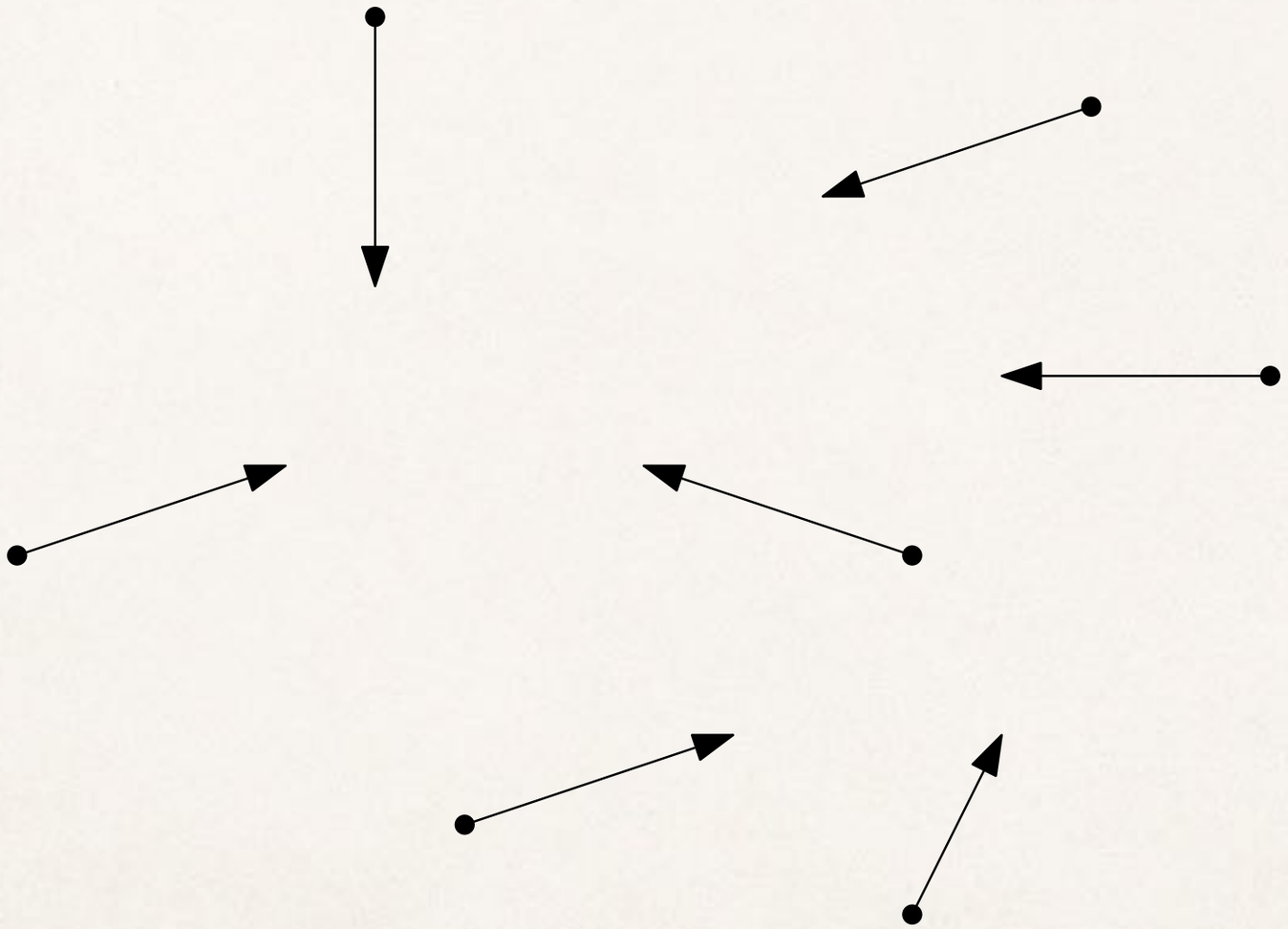


How to use extra space?



- Difficult to walk s steps with a single scan

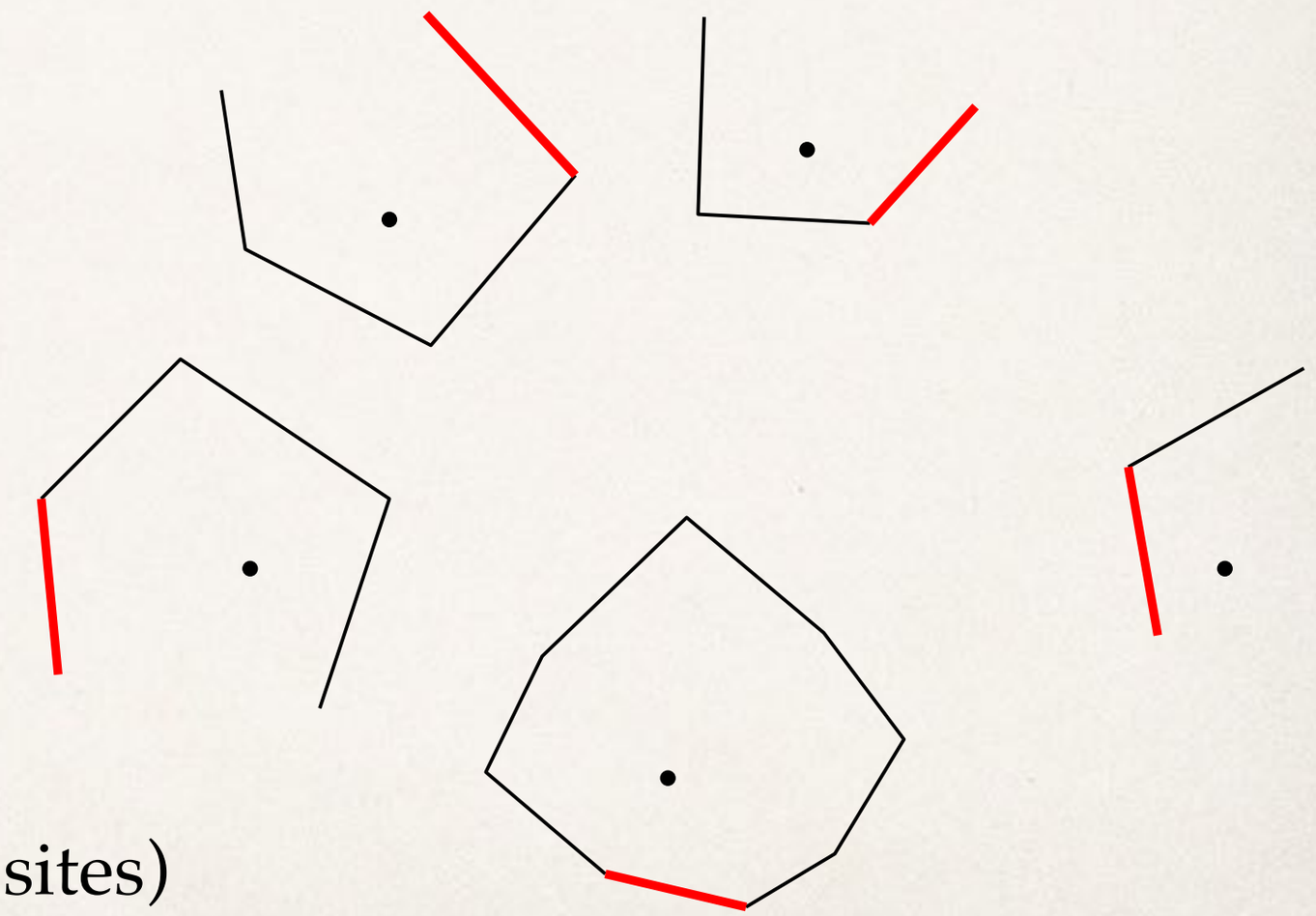
Idea for larger workspaces



Walk one step for s **different** sites

Time-space trade-off algorithm

- Find an edge for all s sites in two scans
  - Discard completed sites
  - Add new sites as needed
- Stop when s sites remain
  - other sites processed
- Only O(s) edges remain (**big** sites)
  - We use *brute force*



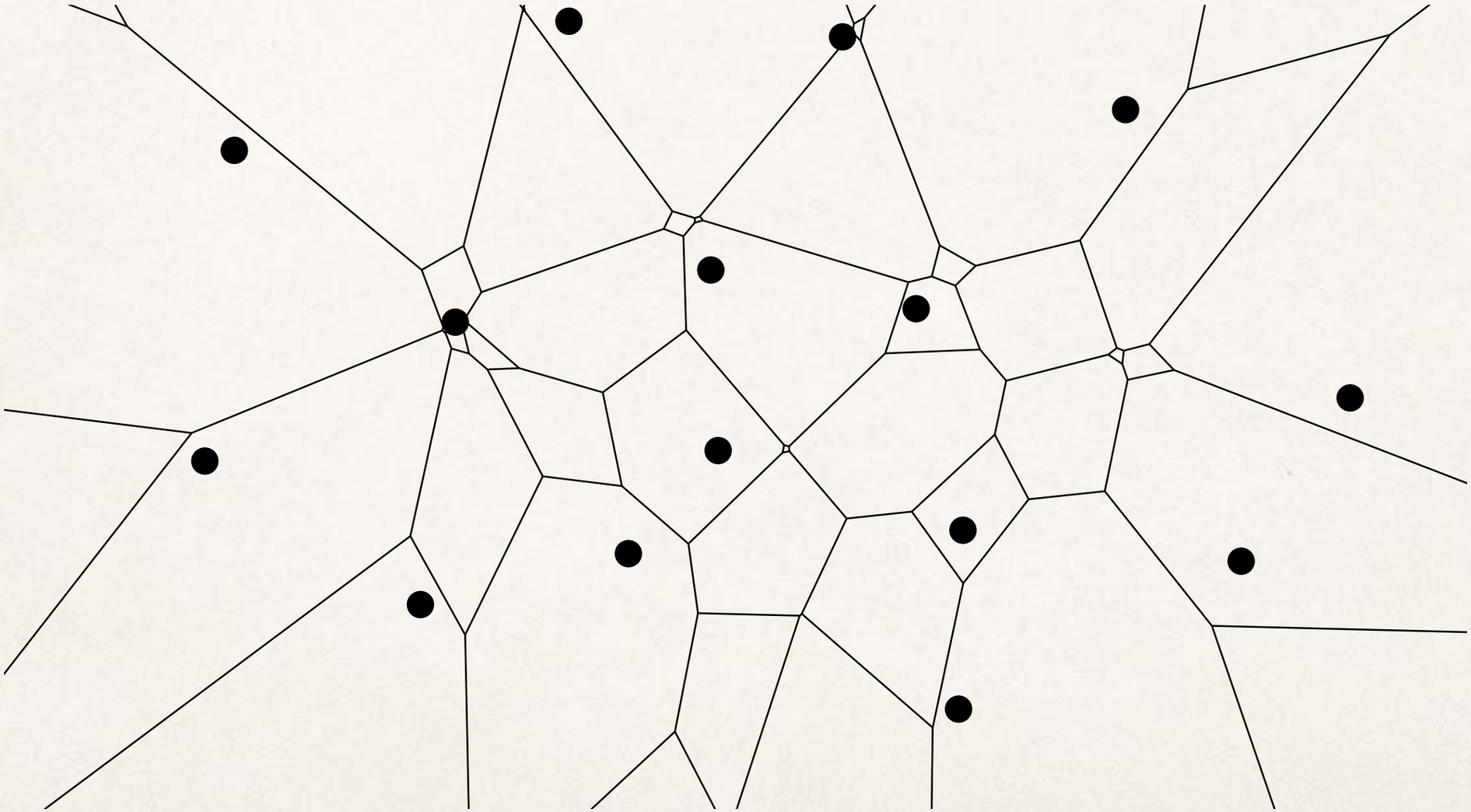
Summary

- Given S sites we can:
  - Compute nearest NVD in $O(n^2)$ time using O(1) words
  - Compute nearest NVD in $O(n^2 \log s/s)$ time using O(s) words
  - Walk along the faces of each cell
    - Same approach works for farthest VD

Summary

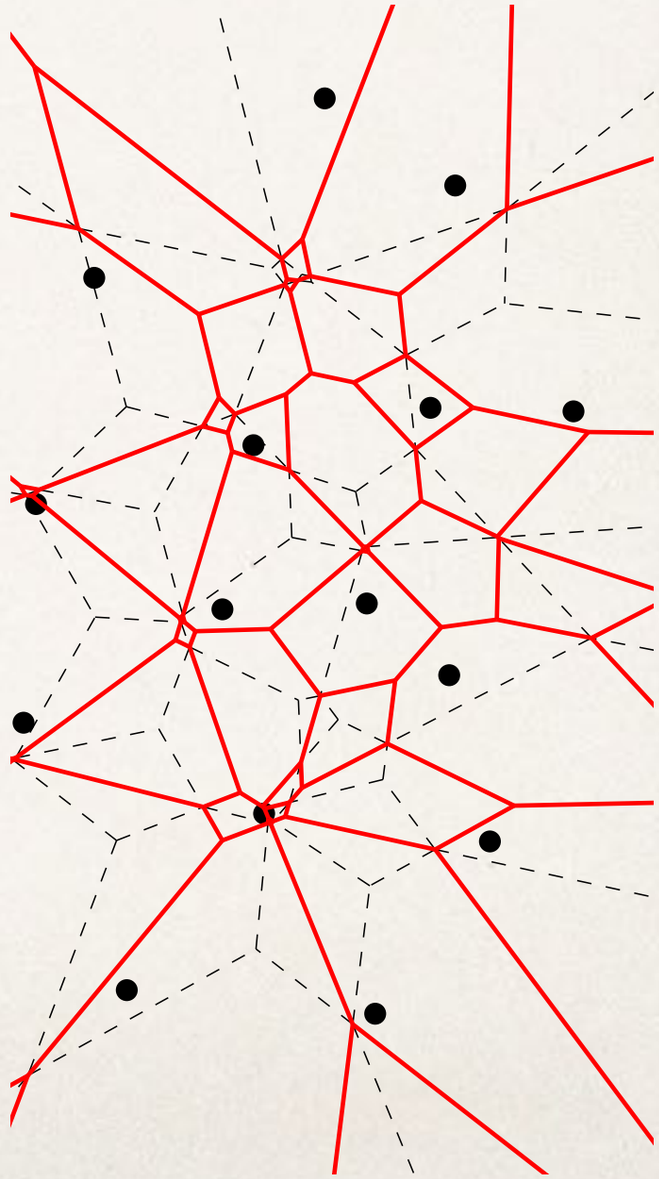
- Given S sites we can:
  - Compute nearest NVD in $O(n^2)$ time using O(1) words
  - Compute nearest NVD in $O(n^2 \log s/s)$ time using O(s) words
  - Walk along the faces of each cell
    - Same approach works for farthest VD
    - **Must** find points in CH(S) first

How to compute high order VD?



Relationship between VD

Look at diagram of order k inside a face of order k+1:



- With a k edge I know the k+1 face I am in
- The order k coincides with FVD
- Edges of order k are either:

  A. Contained in a face of order k+1

  B. Touch a vertex of the face

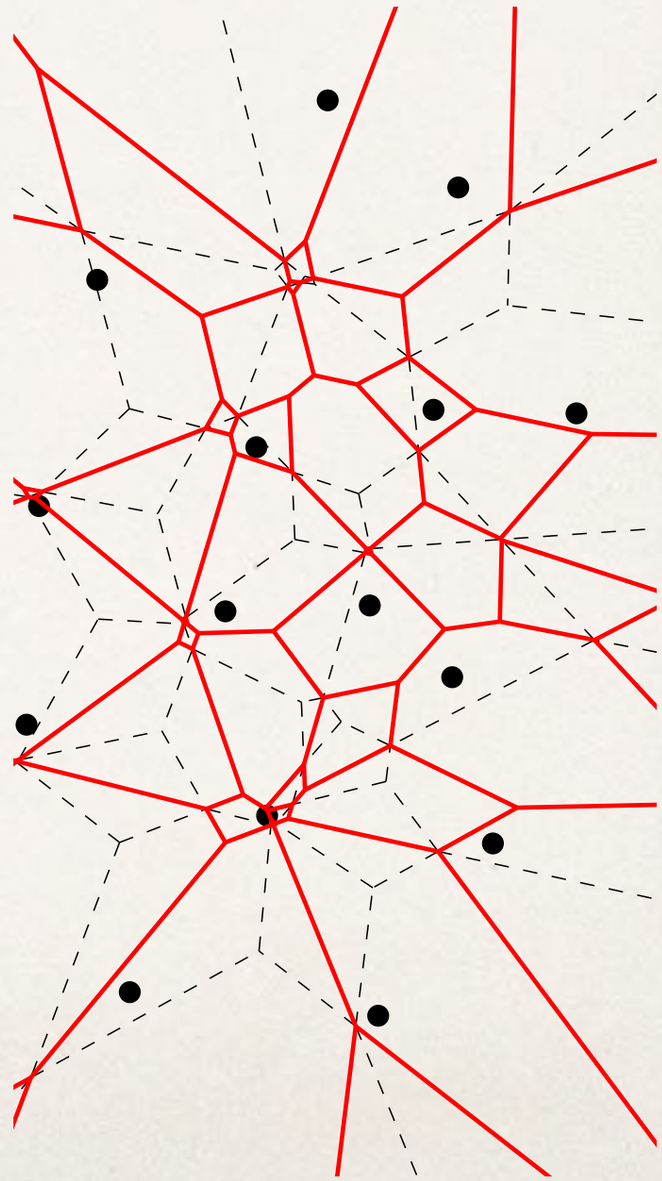
- We can determine which in a single scan

Naive Approach

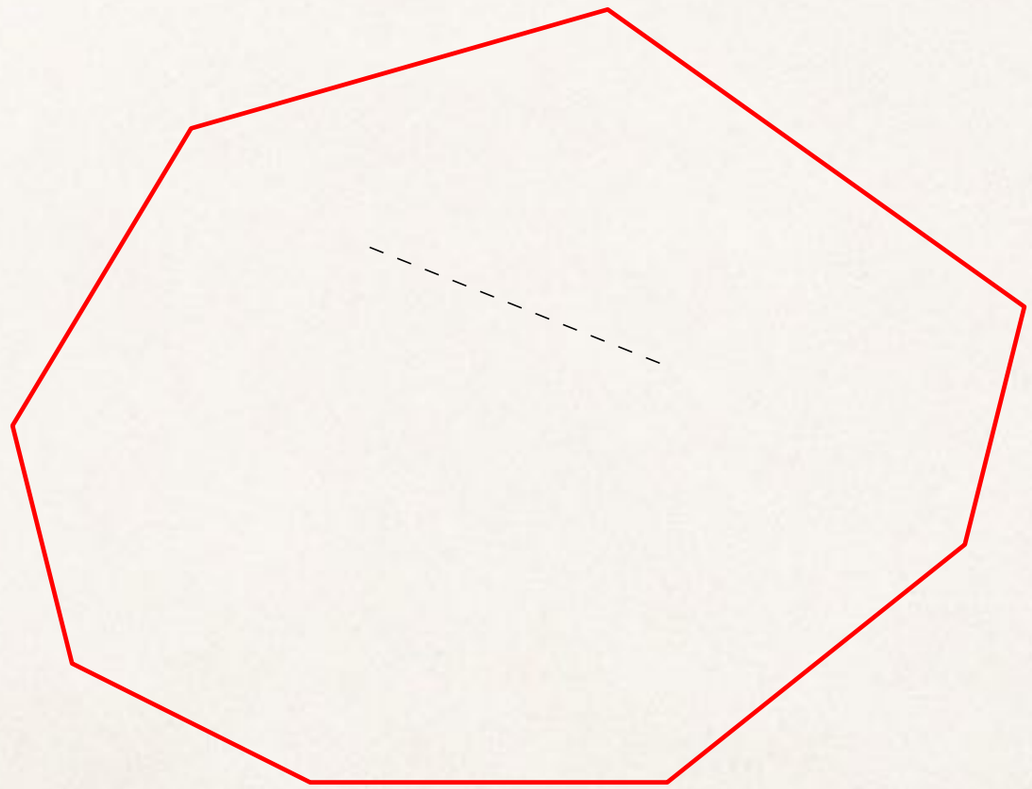
With a k edge we can:

- Find a k+1 edge of the boundary
  - Walk along boundary

  Report the k+1 face it is included
  - Scan the **input** (no need for edges)
- Easy algo: report containing face of k edges
  - O(k!) blowout

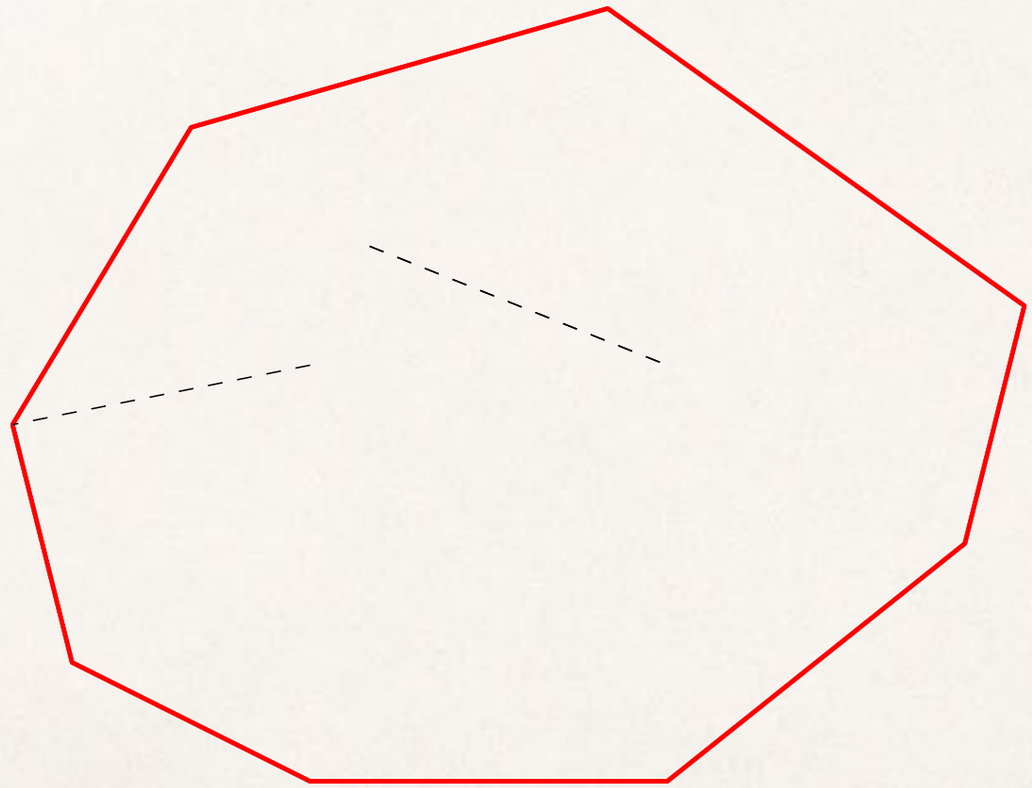


Avoiding Repetitions



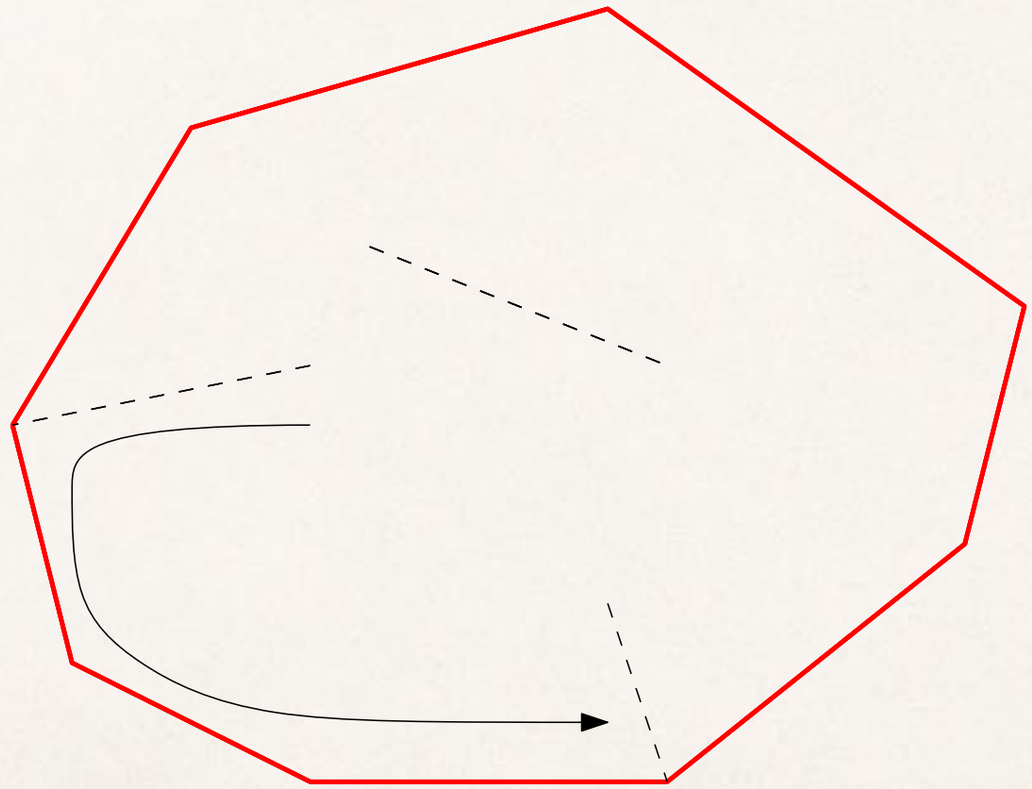
Edges completely **contained** are ignored

Avoiding Repetitions



Edges that touch the boundary **report**

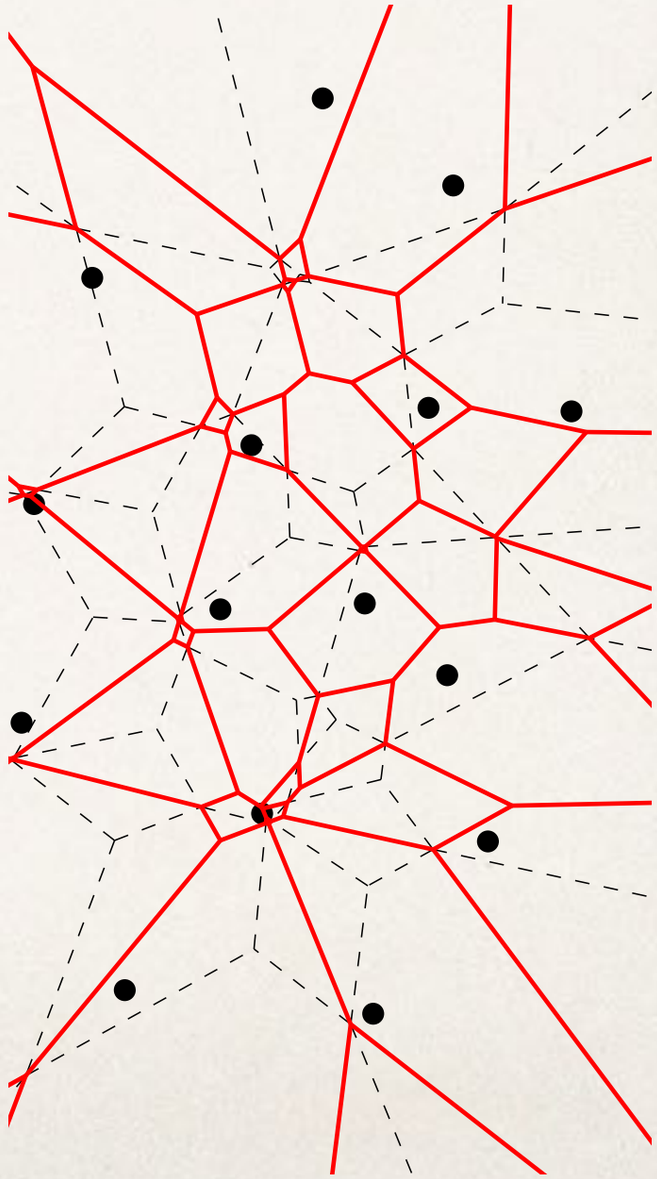
Avoiding Repetitions



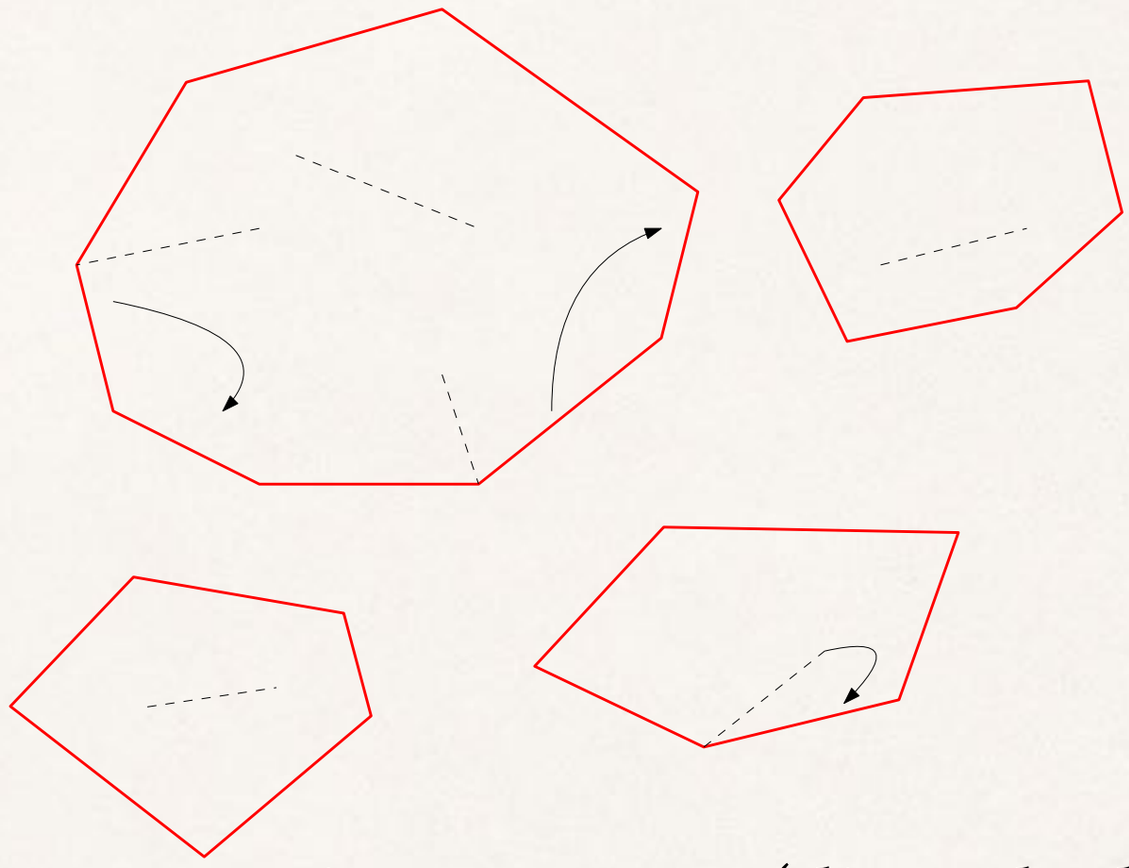
Edges that touch the boundary **report** until **another** boundary edge

Algorithm Sketch

- Store s active order k edges
- While unprocessed edges remain
- Scan S to walk one step of each edge
  - Interior edges are discarded
  - Otherwise, report found edges
- Discard completely reported edges
- Grab additional unprocessed vertices

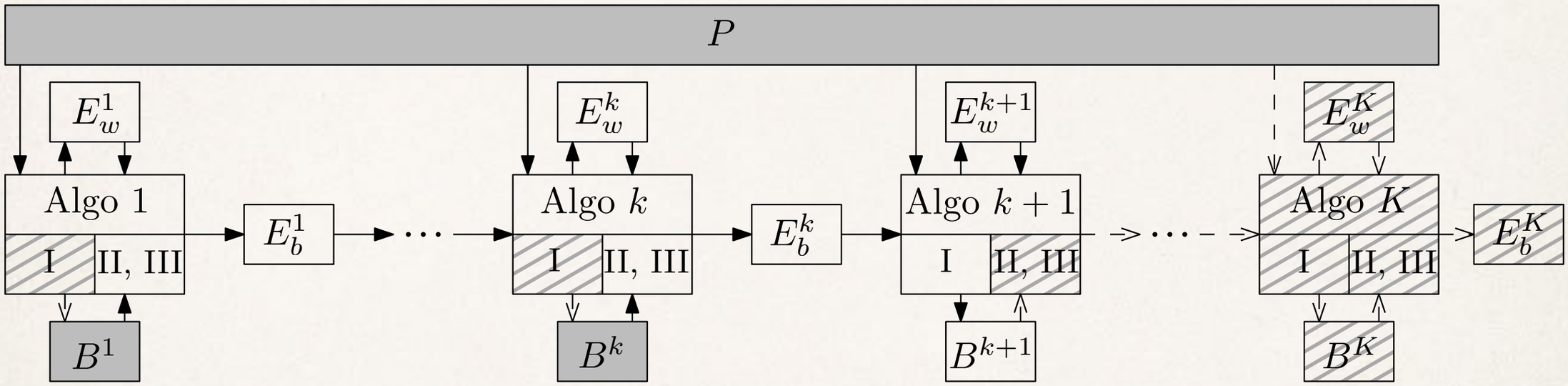


Algorithm Properties



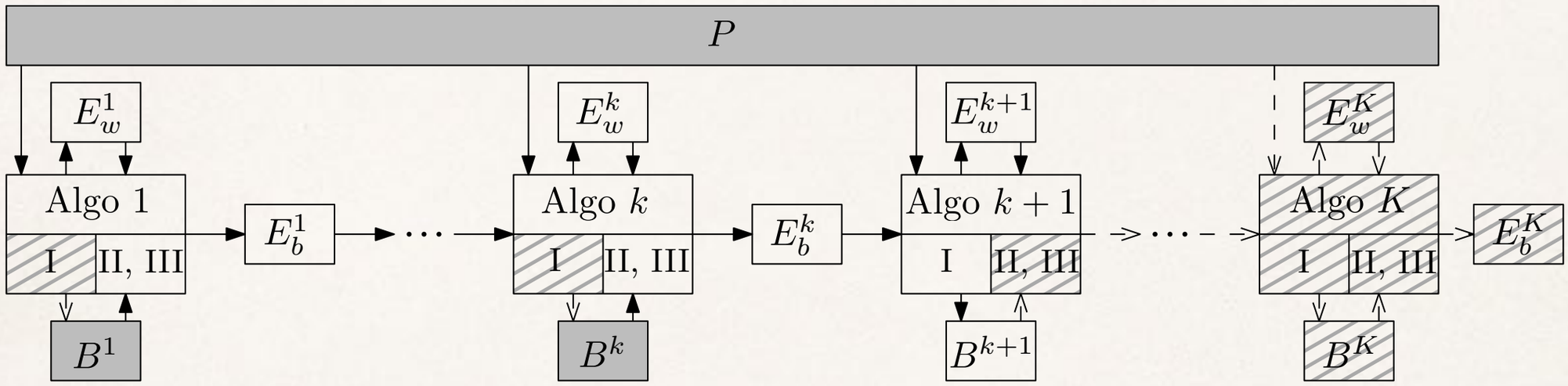
- In one scan of input we gain s **progress** (discard edge/find new one)
  - One scan needs O(nk polylog(k,s)) time
- At most we do O(n/s) scans

Parallel Computation



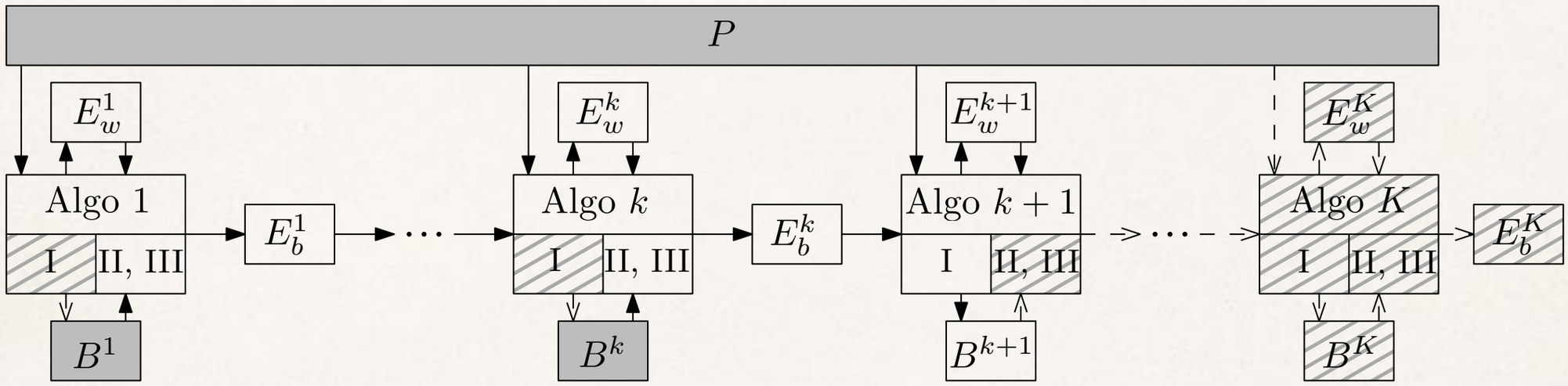
- We cannot store order k edges explicitly
  - Each time we recompute them
  - Each subproblem has own memory of size s’
  - We store between s’ and 3s’ produced edges (then pause)

Several orders at once



- Algorithm runs in **phases**
  - Phase i computes edges of order i+1 from order i
  - Store up to s edges and **pause** the algorithm
    - Edges are consumed by next stage (**resume** when needed)

Time-Space Bounds



- K independent algorithms, each one uses O(is') words
  - Best we can do is $s'=O(s/K^2)$, $K=O(\sqrt{s})$
- K phases, each one needs $O(n^2i^2/s'\ K\ \text{polylog}(k,s))$
  - $O(n^2K^6/s\ \text{polylog}(k,s))$ total time

Summary

- New time-Space trade-off for Voronoi diagrams
  - $O((n^2k^6/s)\ \mathrm{polylog}(k,s))$ using $O(s)$ words
- Improves running time of Nearest VD
  - Remove randomization
- First algorithm for FVD and $k<\sqrt{s}$
- Generalizable to other diagrams (abstract VD?)

Open Problems

- Faster Algorithms/Lower Bounds?
- Compute diagrams of order $\Omega(\sqrt{s})$ and $O(s)$?
- Report edges in increasing order of k?

  Without the additional O(k) factor
- Compute order k diagram without using all small order diagrams?
- Faster algorithms for large s?

Questions? Comments?

Thank you very much for your attention!